Application of Robot Arm in 2D Incision

A project of the 2016 Robotics Course of the School of Information Science and Technology (SIST) of ShanghaiTech University
Instructor: Prof. Sören Schwertfeger · `https://robotics.shanghaitech.edu.cn/teaching/robotics2016`

Cao Yuchen
SIST
caoych@shanghaitech

Zhang Shuai
SIST
zhangshuai1@shanghaitech

Shan Zeyong
SIST
shanzy@shanghaitech

Abstract

We'd like to apply the 6-axes robot arm in traditional 3-axes or 5-axes CNC machine. Traditional 3-axes or 5-axes CNC machine has many limits: 1) Most of traditional CNC machine is in a enclosed box. 2) The milling head just can move in 3-axes and the milling range is limited. 3) The robot arm can move around the object thus the object can be very large, such as a sculpture. The robot arm based CNC machine we build will be capable of doing these job.

***Keywords**: Moveit! Path Planning 2D Incision Driller*

1. Introduction

The Robotic Arm is a programmable mechanical arm, which is mimicing the movement of human arms. The robot arm usually consists of joints and links, the number of different arms may be different. It's common to see a 5 or 6 joints robotic arm. And the links that connect joints make it possible to actuate rotation and translation. And the links are considered as kinematic chain. So we can calculate the speed and pose through different frames relative to base frame. For this project, we choose to use Schunk Arm as showed in Figure 1.

Figure 1. Schunk model

Similar to 3D printer, the idea of 3D incision by robot arm is aimed to carve on a raw material by following determined path to create a 3D model. This requires the program realize avoiding collision, reaching around obstacles and planning divesity of points to arrive at the goal step by step. We thought this is interesting and cool to 3D print something by a robot arm, something that a man's arm even cannot do well.

2. State of the Ar

The algorithm to compute motion planning contains two parts:(a) it utilizes the topology of the arm and obstacles to factor the search space and reduce the complexity of the planning problem using dynamic programming; (b) it takes only polynomial time in the number of joints under some conditions. For the 2D condition, there is a path between two homotopic configurations, an embedded local planner finds a path within a polynomial time. In finding sets of plans that move the position of end-effector, The algorithm first finds the set of accessible positions. For each position, it marks the possible entrance configurations, any angle which links 1 can approach from joint 1 to joint 2 at that position (in this case, the location of joint 1 is fixed in the base, so there is at most one such angle for every position).

For each position for joint 2, we find the set of accessible positions.

ROS stack: moveit. MoveIt is state of the art software for mobile manipulation, incorporating the latest advances in motion planning, manipulation, 3D perception, kinematics, control and navigation. It provides an easy-to-use platform for developing advanced robotics applications, evaluating new robot designs and building integrated robotics products for industrial, commercial, R&D and other domains. The ROS stack moveit contains all essential package of MoveIt!, include moveit_commander, moveit_core, moveit_planners, moveit_plugins,moveit_ros and moveit_setup_assistant. We can create our own moveit package by configuration some essential parameter use move_setup_assistant, and we can also load our arm model in it. Moveit integrates OMPL (Open Motion Planning Library) which contains many motion planning algorithms such as Rapidly-exploring Random Trees (RRT) and Probabilistic Roadmap (PRM).

Reducing prototyping time is a good way to make the product development cycle shorter. This can be achieved in two ways: one is to develop new prototyping technologies like stereolithography apparatus (SLA), selective laser sintering (SLS) etc.; the other is to improve the principal existing technique which is CNC based method. In this paper, a robotic system for rapid prototyping which is an enhancement of the CNC based method is presented. A robot arm holding a milling tool is used to machine the prototype of a solid model drawn in commercial CAD systems. The rough cut and finish cut NC tool path for the robot arm are generated automatically from the solid model of an object. Objects may have different kinds of surfaces like planar surface, general quadratic surface, B-spline surface and compound surface. The proposed method is implemented on the AutoCAD platform. A number of produced prototypes have shown satisfactory results.

ROS Package: TF. TF is a package that let the user keep track of multiple coordinate frames over time. tf maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc between any two coordinate frames at any desired point in time. TF can operate in a distributed system. This means all the information about the coordinate frames of a robot is available to all ROS components on any computer in the system. There is no central server of transform information.

In our robot arm projects. We can use TF package to build every joints of the robot arm. And with TF package we can know the robot arm's transform and control the robot arm better. By listening and broadcasting the transforms, we can make the robot arm interact with other device and process. With transform information, we can optimize the motion of the robot arm.

ROS package: schunk_canopen_driver. This package provides a driver interface for the Schunk LWA4P robot arm through the CANOPEN interface. It provides a simple interface that accepts position commands and another interface for ros_control. The package offers two different interfaces, which both differ in the action topics, parameters and the way commanded waypoints are interpreted by the hardware. The simpler profile_position-interface accepts series of waypoints which the robot will drive to with internal interpolation. It is not guaranteed (and might never happen), that all joints finish moving at the same time. The ros_control-interface provides a position controller in joint space which will interpolate between waypoints inside the controller (on the host PC). You can set joint velocities and time constraints for each waypoint, and the controller will take care. So we are using the ros_control-interface mode.

3. Approach

3.1. Driller and Model Calibration

To simulate the real condition for Schunk arm, first step is to calibrate the model and set the collision attribute. We fisrt describe the urdf and xacro file to build the model for schunk arm in RVIZ and Moveit!. The main work is to write the size, collision area and geometry. Below is a part of prgram of the urdf, each position of link is based on the frame of its parent joint, and one joint may have multiple links, the joint and link both contains the information about their rotation and position. For links, there are shape and collision to describe, to make sure the link can be visualized in model and have collision avoidance attributes. Part of the urdf is listed as below:

```
<!-- joint between arm_6_link and tcp_link-->
<joint name="\${name}\_tcp\_driller\_joint"
type="fixed">
<origin xyz="-0.092 0 0.04" rpy="0 0 0" />
<parent link="\${name}\_tcp\_link"/>
<child link="\${name}\_tcp\_driller"/>
</joint>

<!-- link for driller -->
<link name="\${name}\_tcp\_driller">
<visual>
<origin xyz="0 0 0" rpy="0 1.5708 0" />
<geometry>
<cylinder length="0.08" radius="0.01"/>
</geometry>
<material name="Schunk/DarkGrey" />
</visual>
<collision>
<origin xyz="0 0 0" rpy="0 1.5708 0" />
<geometry>
<cylinder length="0.08" radius="0.01"/>
```

```
</geometry>
</collision>
</link>
```

After many times recorrect data, we finally get a relative similar model with the real one. And then, we use 3D printer to design the model to fix driller on Schunk arm. The desgin model is drawn in Solidworks, shown as below:

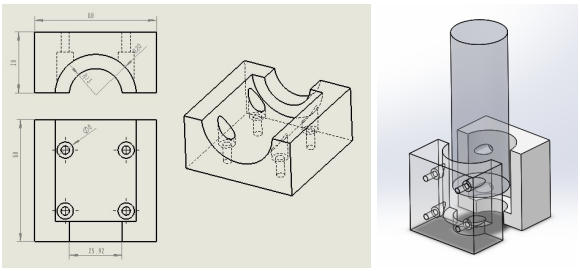

Figure 2. Fixing Model.

3.2. Generate the Coordinates for Carving

We choose the PyCAM software to generate the path of 2D/3D incision. The software goes like below:

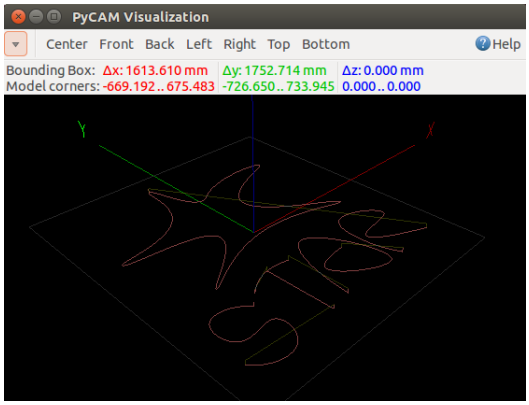

Figure 3. PyCAM.

In this window, we can generate any shape of model as we want, it provides the reserved shape and also allow us to draw it by ourselves. Here we choose a simple pentaon as the goal path points. And we can get the raw datas like this:

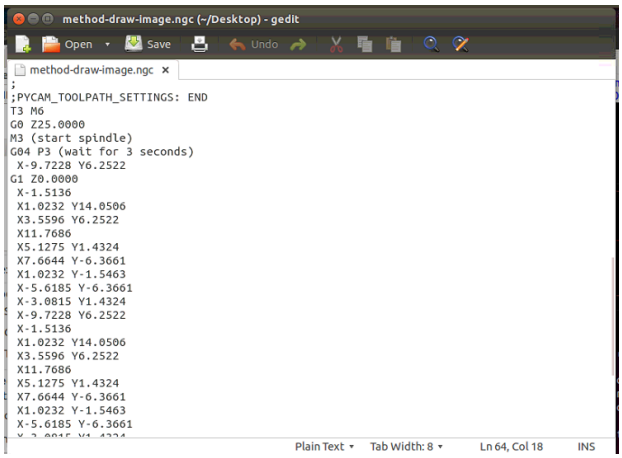

Figure 4. Path points

Then, we write a reading data program by C++. During compiling, the data type convertion and the loop time is important factor to consider. First time, we choose to generate a bag and let main program to listen to this topic, but it takes extra time, so we finally combine the reading program with main program together to save effort. The main program goes like:

```
ifstream in("/home/ubuntu/schunk/src/schunk_read/src
/pycam-text.ngc",ios::binary);
while (!in.eof())
{ in >> s1;
if (s1[0] == 'X')
{ if (s2[0] == s1[0])
{if (first)
{ xp = x, yp = y, zp = z;
first = false;}
target_pose3.position.x += (x-xp)/100.0;
target_pose3.position.y += (y-yp)/100.0;
target_pose3.position.z += (z-zp)/100.0;
waypoints.push_back(target_pose3);
xp = x, yp = y, zp = z; }
sx = &s1[1];
x = strtod(sx, NULL); }
else if (s1[0] == 'Y')
{ sy = &s1[1];
y = strtod(sy, NULL);
if (first)
{ xp = x, yp = y, zp = z;
first = false;
} target_pose3.position.x += (x-xp)/100.0;
target_pose3.position.y += (y-yp)/100.0;
target_pose3.position.z += (z-zp)/100.0;
waypoints.push_back(target_pose3);
xp = x, yp = y, zp = z; }
else if (s1[0] == 'Z')
{ sz=&s1[1];
```

z = strtod(sz,NULL); }
else ;
s2 = s1; }
in.close();

With the help of this program, we input the data into program and make complicated path planning available.

3.3. Move the Arm by Moveit!

MoveIt! is state of the art software for mobile manipulation, incorporating the latest advances in motion planning, manipulation, 3D perception, kinematics, control and navigation. It provides an easy-to-use platform for developing advanced robotics applications, evaluating new robot designs and building integrated robotics products for industrial, commercial, R&D and other domains. To use Moveit!, first we need to create our own package through MoveIt! Setup Assistant. It can be start by command: roslaunch moveit_setup_assistant setup_assistant.launch This will bringup the start screen with two choices: Create New MoveIt! Configuration Package or Edit Existing MoveIt! Configuration Package.

Figure 5. Moveit!

We can load our schunk arm URDF file, in which LWA4P is provided from schunk CANopen driver package, and download schunk pg70 gripper. Also, we have done some change in it, that is add the camera, table and drill.

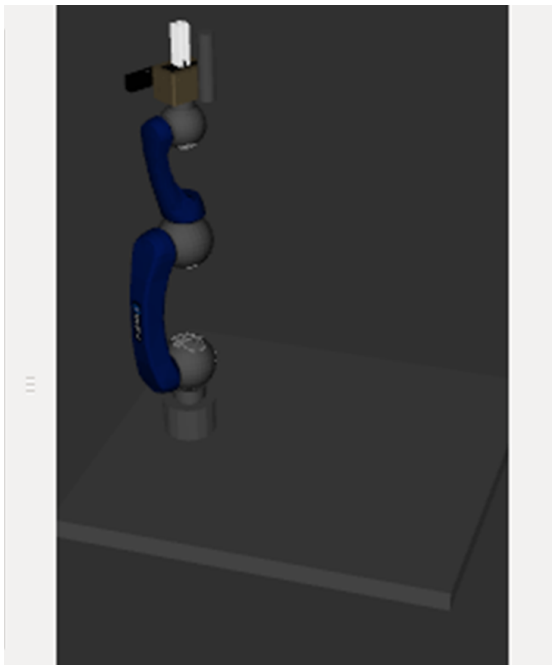

Figure 6. Moveit! model

Then, we need to generate self-collision matrix, because when planing, some link pairs are no need to calculate the self-collision. For example, Arm_0_link and arm_2_link is disabled because they will never in collision:

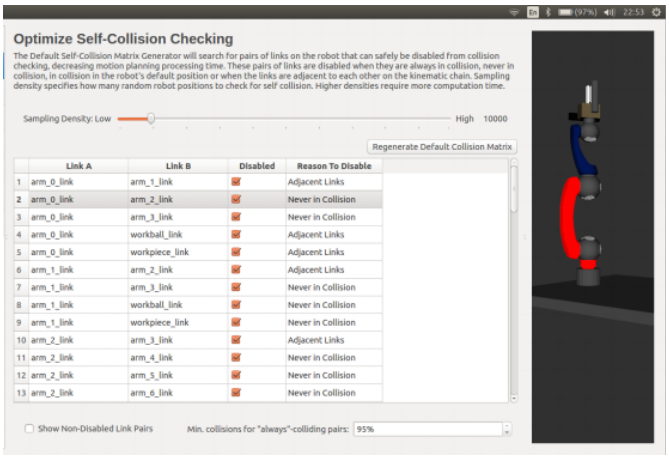

Figure 7. Moveit! Collision Arm

Workball_link and workpiece_link is disabled because they are adjacent links. Then we create a virtual joint called world_joint, virtual joints are used primarily to attach the robot to the world. Next is the most important part, we need to set our move group, we set the group name to Arm, and select the kinematics solver, you can even write your own kinematics solver.

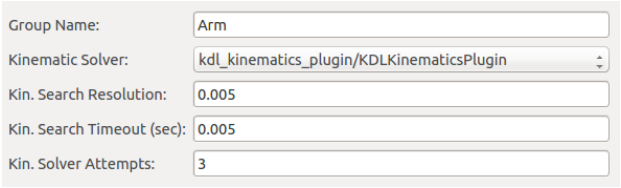

Figure 8. Moveit! Kinematics Solver

Our planning group is like figure 9:

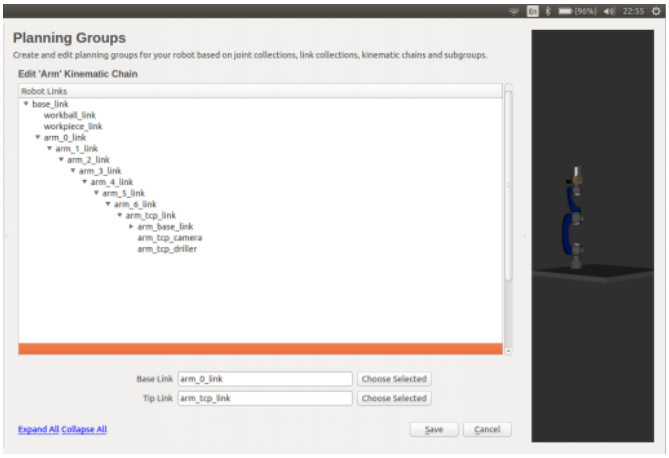

Figure 9. Moveit! Planning Group

And then we can set some robot poses group, which will be convenient later. And select our end-effecor. The end-effector is important because its pose is just the x,y,z we want the arm to go. Then generate the package. MoveIt! operates on sets of joints called planning groups and stores them in an object called the JointModelGroup. Throughout MoveIt! the terms planning group and joint model group are used interchangably.

static const std::string PLANNING_GROUP = "Arm";

The MoveGroup class can be easily setup using just the name of the planning group you would like to control and plan for.

moveit::planning_interface::MoveGroupInterface move_group(PLANNING_GROUP);

We will use the PlanningSceneInterface class to add and remove collision objects in our virtual world scene.

moveit::planning_interface::PlanningSceneInterface planning_scene_interface;

We can print the name of the reference frame for this robot.

ROS_INFO_NAMED("tutorial", "Reference frame: %s", move_group.getPlanningFrame().c_str());

We can also print the name of the end-effector link for this group.

ROS_INFO_NAMED("tutorial", "End effector link: %s", move_group.getEndEffectorLink().c_str());

We can plan to a pose goal, which means we set a x,y,z and p,r,y, and execute the plan, we can also plan to a joint-space goal. But what we need in this project is cartesian path plan, because plan to a pose goal can guarantee that the trajectory is what we want, since the planner plans in the joint space, the shortest path in the joint space is always not the shortest in the world space.

We can plan a cartesian path directly by specifying a list of waypoints for the end-effector to go through. Note that we are starting from the new start state above. The initial pose (start state) does not need to be added to the waypoint list but adding it can help with visualizations.

std::vector geometry_msgs::Pose waypoints;
waypoints.push_back(start_pose);
geometry_msgs::Pose target_pose = start_pose;
target_pose.position.z+=0.2;waypoints.push_back(target_pose);
target_pose.position.y-=0.1;waypoints.push_back(target_pose);
target_pose.position.z-=0.2;target_pose3.position.y+=0.2;
target_pose.position.x-=0.2;waypoints.push_back(target_pose);

Cartesian motions are frequently needed to be slower for actions such as approach and retreat grasp motions. Here we demonstrate how to reduce the speed of the robot arm via a scaling factor of the maxiumum speed of each joint. Note this is not the speed of the end effector point.

move_group.setMaxVelocityScalingFactor(0.1);

We want the cartesian path to be interpolated at a resolution of 1 mm which is why we will specify 0.001 as the max step in cartesian translation. We will specify the jump threshold as 0.0, effectively disabling it. Warning - disabling the jump threshold while operating real hardware can cause large unpredictable motions of redundant joints and could be a safety issue.

moveit_msgs::RobotTrajectory trajectory;
const double jump_threshold = 0.0;const double eef_step = 0.01;
double fraction = move_group.computeCartesianPath(waypoints, eef_step, jump_threshold, trajectory);
*ROS_INFO_NAMED("tutorial", "Visualizing plan 4 (cartesian path) (%.2f%% acheived)", fraction * 100.0);*

And the function will automatically visualize the plan in Rviz. If the visualize is in our expectation, we can execute the plan.

4. Experiment and Result

Our system is maily aimed to build a robot-arm based 5 or 6 axes CNC machine. It consists of 3 parts:

1.Model building: We need to build up mathematica model to show the trasformation and rotation among different joints.

2.Motion planning and g-code generate: we try to build up a bridge to take the use of G-code, which is highly integrated and easily to control 3D printer, so that the

motion planning can be realized at the same time.
3.Arm execute: just apply it and check and recorrect the error!

We finally adjust the algorithm to make sure that everytime the exectution is 100% successful, and there is no missing point or path in pratice. However, because of the limitation of velocity, Schunk arm does not completely goes in average speed as we expect. Here is an example we draw by schunk arm:

Figure 10. Example for 2D Drawing.

As we can see, the result is not bad, and the accuracy is high when the driller rotating speed is quick enough and the board is not so tough, i.e. the friction force is too large. Otherwise the arm will departure from the planning path with strong friction.

5. Time-Line

03/11/2016-05/11/2016
Research, read paper, and wrote proposal for our project.
14/11/2016-20/11/2016
Generated the coordinates for path planning, made sample program to motivate the arm do some simple movements, and wrote the reading txt program to get the coordinates.
21/11/2016-27/11/2016
Wrote the Xtmls to describe a model with visual and colliding attributes, finally we drive the Moveit! to draw some squres and ellipse by robot arm.
05/12/2016-11/12/2016
Solved bugs that appear in our trying, 3D printed the pedestal to install the driller on Robot Arm.
12/12/2016-18/12/2016
Planned complicated path for the arm and got prepared for mid-term presentation.
19/12/2016-10/1/2016
Fix bugs such as getting connected with finger of gripper which is out of control, and refine the algorithm so that it can draw any 2D pictures as long as it stays in its touchable domain.
11/1/2016-19/1/2016
And getting prepared for final report.

6. Conclusions

In conclusion, our team firstly focused on 2D incision, and the main problem was to build a mathematical model, which contains relationship between different frames. Then based on each servo motors' rotating ability, we applied the model and motion planning method into program, then we kept repeating test till it works well. In those processes, we three students learned the knowledge about ROS system, Robot Arm and how to program to manage path planning. Also, we develop the ability of team work.

Finally, we will thank professor Soren Schwertfeger for giving us much help in our project, giving us a very nice introduction to the robot arm, and providing us with so nice hardware Schunk, Xtion and ohter equipments. And he patiently teaches us a lot from scratch. We will also thank our University for providing us so nice place to study and do experiments.

References

[1] H. Choset. Robotic motion planning: Configuration space. *Robotics Institute 16-735.*

[2] D.-H. L. J. Y. L. Dong-Hyung Kim, Sung-Jin Lim and C.-S. Han. A rrt-based motion planning of dual-arm robot for (dis)assembly tasks. *DOI: 10.1109/ISR.2013.6695698. IEEE. 06 January 2014.*

[3] E. A. Jaesik Choi. Factor-guided motion planning for a robot arm. *University of Illinois at Urbana-Champaign. Urbana, IL 61801. Oct 29 - Nov 2, 2007.*

[4] J. W. Richard Tatum, Drew Lucas and J. Perkins. Geometrically motivated inverse kinematics for an arm with 7 degrees of freedom. *IEEE.*

[5] G. M. Sergey Pluzhnikov. Motion planning and control of robot manipulators. *Norwegian University of Science and Technology. June 2012.*

[6] M. S. Tobias Kunz, Ulrich Reiser and A. Verl. Real-time path planning for a robot arm in changing environment. *IEEE/RSJ International Conference on Intelligent Robots and Systems. (IROS'10) Oct. 2010.*

[7] Y. C. WC Tse. A robotic system for rapid prototyping. *Proceedinps of the 2000 IEEE. lntemahonal Conference on Robotics & Automation. San Francisco. CA April 2000.*

[8] Y. C. YN Hu. Implementation of a robot system for sculptured surface cutting. department of mechanical engineering. *The University of Hong Kong, Hong KongCN. DOI: 10.1007/s001700050112.*

[9] B. X. J. Zhao and Y. Liu. Human-like motion planning for robotic arm system. *DOI:10.1109/ICAR.2011.6088543. IEEE. 01 December 2011.*